Pusan National University
Computer Science and Engineering
Technical Report 2025-09

Simulink 기반
자동차 부품 연계보안 기술 연구

202029178 홍재왕
202055627 레퐁푸
202055555 석재영

지도교수 손준영

목 차

1. 서론

1.1 연구 배경

현대 자동차 산업은 자율주행, 커넥티드카, 전기차 기술의 급격한 발전으로 전례 없는 전자화 시대를 맞이하고 있다. 차량 내부에는 수십 개의 전자제어장치(ECU), 센서, 액추에이터가 탑재되어 있으며, 이들은 CAN(Controller Area Network), LIN, Automotive Ethernet 등의 통신 프로토콜을 통해 실시간으로 데이터를 교환하며 복잡한 제어 기능을 수행한다. 특히, 차량-외부 통신(V2X)과 클라우드 기반 서비스의 확산은 차량 네트워크의 연결성을 더욱 강화하고 있다.

그러나 이러한 고도화된 통신 환경은 새로운 보안 위협을 동반한다. CAN 프로토콜의 낮은 보안성, Ethernet 기반 통신의 외부 노출 가능성 등으로 인해 해킹, 데이터 스푸핑, 악성코드 주입, 리플레이 공격 등의 위험이 증가하고 있다. 실제로, 2015년 Jeep Cherokee 원격 해킹 사건과 같은 보안 사고는 자동차 보안의 취약성을 전 세계적으로 드러냈다. 이러한 위협은 운전자와 승객의 안전뿐만 아니라 자동차 제조사의 신뢰도와 산업 전반에 심각한 영향을 미칠 수 있다.

이에 본 과제는 MathWorks의 Simulink 플랫폼을 활용하여 자동차 전장부품 간 연계 동작을 모델링하고, 부품별 특성에 최적화된 보안 알고리즘(예:AES, ECC, HMAC)을 설계·구현하는 것을 목표로 한다. 다양한 보안 알고리즘의 성능과 적용 가능성을 시뮬레이션 환경에서 비교·검증함으로써, 사용자에게 최적의 보안 대안을 선택할 수 있는 체계적인 프레임워크를 제공하고자 한다. 이는 자동차 보안 기술의 신뢰성을 높이고, 향후 자율주행차 및 커넥티드카의 안전한 상용화에 기여할 것이다.

1.2 기존 문제점

가. CAN 프로토콜의 보안 한계: CAN은 경량성과 실시간성에 최적화되어 있으나, 인증·암호화 기능이 없어 데이터 위변조·리플레이 공격에 취약하다.

나. Ethernet 통신의 외부 노출 위험: 고속·대용량 전송이 가능하지만, 외부 네트워크와 직접 연결되기 때문에 해킹·침입 가능성이 높다.

다. 보안 알고리즘 적용의 복잡성: AES, ECC, HMAC 등 다양한 보안 기법이 존재하나, ECU 자원 제약(CPU, 메모리) 및 실시간 제약으로 인해 직접적인 적용이 어렵다.

라. 실험적·시뮬레이션 환경 부족: 실제 차량 환경에서 보안 실험은 비용과 위험이 커, 보안 알고리즘을 쉽게 비교·검증할 수 있는 체계적 시뮬레이션 프레임워크가 부재하다.

1.3 연구 목표

본 프로젝트의 궁극적인 목표는 MATLAB/Simulink 환경에서 CAN-FD 기반 자동차 전장부품 데이터 전송 시스템을 설계하고, 여기에 보안 알고리즘(HMAC-SHA256, Freshness Counter)을 적용하여 실제 차량 내부 통신 환경에서 발생할 수 있는 위협을 모의 검증하는 것이다. 이를 위해 송신 ECU에서 생성된 데이터(AppData, Freshness)가 수신 ECU에서 동일한 MAC 연산을 통해 검증되도록 하여, 데이터 위변조 및 재전송 공격을 차단하고, 정상적으로 인증된 데이터만이 제어 시스템에 입력되는 보안 게이트 구조를 구현한다.

특히, 본 연구는 CAN-FD 통신의 특성과 제약을 고려하여, 무결성 검증(HMAC-SHA256)과 신선도 검증(Freshness Counter)을 동시에 적용하는 메커니즘을 설계함으로써, 기존 CAN 환경에서 부족했던 보안성을 강화한다. 이 과정을 통해, 공격 시나리오(위변조, 재전송, MAC 불일치)에 대한 ECU 동작을 실험적으로 검증하고, 보안 모듈의 민감성 및 동기화 필요성을 체계적으로 분석한다.

또한, 기존의 단일 프로토콜 기반 보안 검증에서 나아가, 본 프로젝트는 Ethernet 통신 경로를 병렬로 구현하여 CAN-FD와 함께 운용할 수 있는 시뮬레이션 환경을 마련한다. Ethernet 기반 전송은 고속·대용량 데이터 처리에 강점을 가지는 반면, 외부 네트워크와의 연결로 보안 위협이 크다는 특성을 지닌다. 따라서 본 연구에서는 CAN-FD와 Ethernet 경로를 동시에 모델링하고, 동일한 보안 알고리즘을 병렬 적용하여, 차세대 차량 네트워크에서의 다중 프로토콜 환경 보안성 검증을 목표로 한다.

결국 본 프로젝트는 Simulink 시뮬레이션 환경에서 CAN-FD와 Ethernet을 통합한 보안 프레임워크를 제시하고, 이를 통해 다양한 보안 알고리즘의 적용 가능성을 실험적으로 검토함으로써, 향후 자율주행차 및 커넥티드카 환경에서 신뢰성 높은 ECU 보안 구조를 구현할 수 있는 기반을 마련하고자 한다.

2. 연구 배경

2.1 Simulink

Simulink는 MathWorks에서 제공하는 모델 기반 설계(Model-Based Design, MBD) 플랫폼으로, 복잡한 시스템의 동작을 시각적으로 설계, 시뮬레이션, 검증할 수 있는 강력한 환경을 제공한다. 본 프로젝트에서는 Simulink를 활용하여 자동차 전장부품(ECU, 센서, 액추에이터)과 통신 프로토콜(CAN, Ethernet)을 모델링하고, 보안 알고리즘을 통합한 가상 시뮬레이션 환경을 구축한다. 이를 통해 부품 간 연계 동작과 보안 시나리오를 테스트하며, 다양한 알고리즘의 성능을 비교·분석한다. 학습 내용은

Simulink 환경 구축과 시뮬레이션 워크플로우 설계로 구분된다.

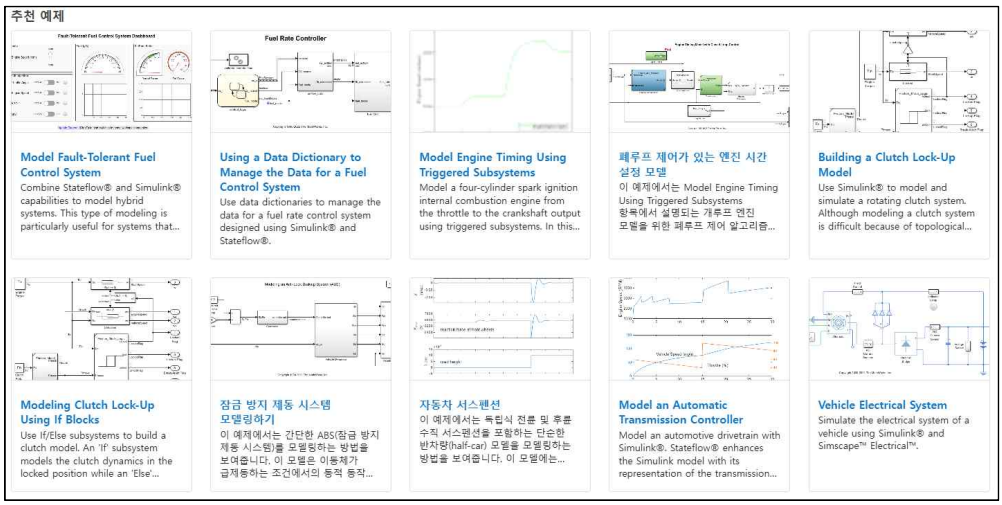


FIG1. Simulink Library

2.1.1 Simulink 환경 구축

Simulink 환경은 차량 내 전장부품과 통신 시스템을 시뮬레이션하기 위한 기반으로 설정되었다. Simulink Library Browser를 활용하여 Vehicle Network Toolbox, Control System Toolbox 등의 라이브러리에서 블록을 선택해 모델을 구성했다. 전체 시스템은 다음과 같은 흐름으로 설계되었다: 센서 데이터 생성 → 암호화 → CAN/Ethernet 전송 → 복호화 → 출력 시각화. 각 전자제어장치(ECU)는 Subsystem 블록으로 모듈화하여 실제 차량의 부품 구조를 반영했으며, 마스크 아이콘을 설정해 블록 간 시각적 구분을 명확히 했다. 예를 들어, 센서 ECU는 데이터 생성 및 전송을, 제어 ECU는 데이터 수신 및 처리를 담당하도록 구성했다. 또한, Signal Builder와 Constant 블록을 사용해 다양한 입력 시나리오를 생성하며, Simulink Configuration Parameters를 조정해 실시간 시뮬레이션 성능을 최적화했다.

2.1.2 시뮬레이션 워크플로우 설계

시뮬레이션 워크플로우는 차량 내 ECU 간 데이터 송수신과 보안 알고리즘 적용 과정을 반영하여 체계적으로 설계되었다. 워크플로우는 다음과 같은 단계를 포함한다:

(1) 암호화 및 복호화 구현: MATLAB Function 블록을 활용해 AES, SPECK 등 경량 보안 알고리즘을 구현했다. 입력 데이터(예: 센서 값)를 암호화하고, 수신 ECU에서 복호화하는 과정을 시뮬레이션했다.

(2) 부품별 단위 테스트: 각 ECU의 Subsystem을 독립적으로 테스트하여 정상 동작을 검증했다. 예를 들어, CAN 통신 블록을 통해 데이터 전송의 무결성을 확인하고, 오류 발생 시 Diagnostic Viewer로 디버깅했다.

(3) 신호 흐름 구조화: Bus Creator와 Bus Selector를 사용해 복잡한 신호 흐름을 구조화했으며, 데이터 송수신 순서를 명확히 정의했다. 이는 CAN의 우선순위 기반 전송과 Ethernet의 고속 통신 특성을 반영한다.

(4) 결과 분석 및 비교: 보안 알고리즘 적용 여부에 따른 성능(지연 시간, 리소스 사용)을 비교하기 위해 To Workspace 블록으로 데이터를 MATLAB 워크스페이스에 저장하고, Scope 및 Display 블록으로 실시간 시각화했다. 또한, MATLAB Script를 연동해 정량적 분석(예: 처리 속도, 메모리 소비)을 수행했다.

다. 프로젝트와의 연계: Simulink 학습은 프로젝트의 핵심 목표인 부품별 보안 최적화와 알고리즘 비교를 위한 기반을 제공한다. Simulink를 통해 CAN/Ethernet 통신 환경을 가상으로 구현하고, 스푸핑, 리플레이 공격 등 보안 위협 시나리오를 시뮬레이션한다. 이를 바탕으로 보안 알고리즘의 실시간 성능과 하드웨어 제약 적합성을 평가하며, MATLAB App Designer로 개발된 UI와 연동해 결과를 시각화한다. 학습 결과를 통해 팀은 Simulink의 모듈화 설계, 통신 시뮬레이션, 데이터 분석 기능을 익혔으며, 이는 후속 단계인 부품 모델링과 알고리즘 통합에 직접 활용된다.

2.2 CAN(Controller Area Network) 통신 프로토콜

2.2.1 CAN이란?

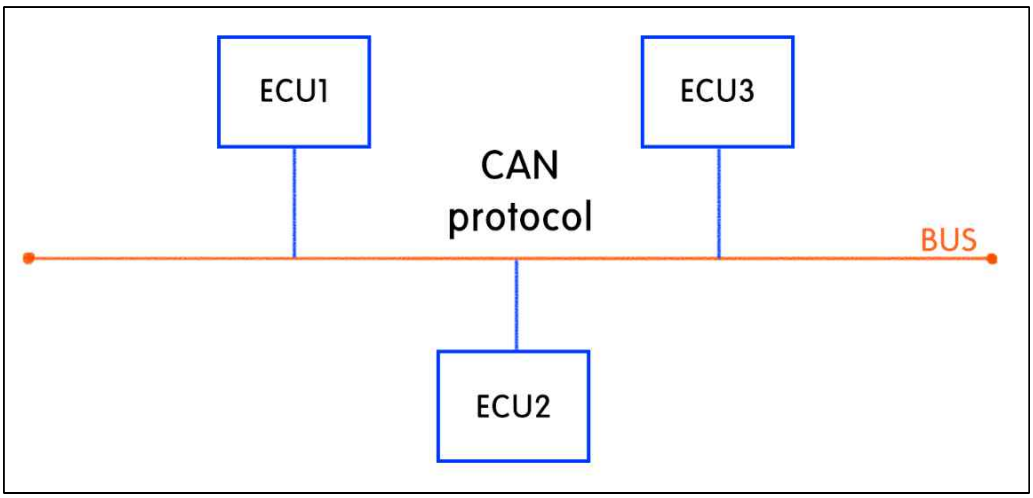


FIG2. CAN Protocol

Controller Area Network(CAN) 프로토콜은 자동차 내 ECU(Electronic Control Unit), 센서, 액추에이터 간 실시간 데이터 교환을 위한 표준 통신 프로토콜로, 본 프로젝트에서 핵심적인 연구 대상이다. CAN은 1980년대 Bosch에 의해 개발되었으며, 낮은 비용, 높은 신뢰성, 실시간성을 바탕으로 자동차 및 산업 제어 시스템에서 널리 사용된다. 본 학습에서는 CAN 프로토콜의 구조, 동작 원리, 보안 취약점, 그리고 Simulink를 활용한 모델링 방안을 분석했다.

2.2.2 CAN 프로토콜의 구조 및 특징

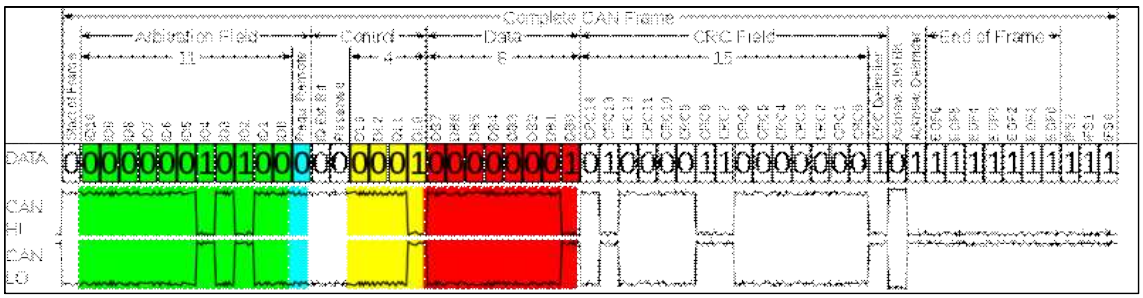


FIG3. 표준 CAN 프레임 포멧

CAN은 멀티마스터, 메시지 중심의 통신 프로토콜로, 최대 1Mbps의 속도를 지원한다. 데이터 프레임은 식별자(Identifier), 데이터 필드(최대 8바이트), CRC(순환중복검사) 등으로 구성되며, CSMA/CR(충돌 해결 방식)을 통해 우선순위 기반으로 메시지를 전송한다. 이는 실시간 제어에 적합하지만, 대역폭 제한으로 인해 보안 데이터를 추가하기 어렵다. CAN은 브로드캐스트 방식으로 동작하므로, 모든 노드가 메시지를 수신하며, 노드 인증이나 암호화 기능이 내장되어 있지 않다.

2.2.3 보안 취약점

CAN 프로토콜은 보안 메커니즘이 부재하여 여러 위협에 노출된다. 첫째, 메시지 위조 공격은 악의적인 노드가 유효한 식별자를 사용하여 거짓 데이터를 전송하는 것으로, 예를 들어 브레이크 제어 신호를 교란할 수 있다. 둘째, 리플레이 공격은 이전에 전송된 메시지를 재전송하여 시스템을 오작동시키는 방식이다. 셋째, DoS(Denial of Service) 공격은 높은 우선순위의 메시지를 지속적으로 전송하여 네트워크를 마비시킬 수 있다. 이러한 취약점은 2015년 Jeep Cherokee 해킹 사건에서 CAN 네트워크를 통한 원격 제어 가능성을 보여주며, 자동차 보안의 시급성을 부각시켰다.

2.2.4 프로젝트와의 연계

본 프로젝트는 CAN 기반 통신 환경에서 보안성을 강화하는 것을 목표로 한다. 이

를 위해, CAN 프로토콜의 메시지 구조와 동작을 Simulink의 Vehicle Network Toolbox를 활용해 모델링한다. 예를 들어, ECU 간 데이터 교환을 시뮬레이션하고, 메시지 인증 코드(MAC)나 경량 암호화 알고리즘(예: SPECK)을 적용하여 데이터 무결성과 기밀성을 보장하는 방안을 테스트한다. 또한, CAN의 제한된 대역폭을 고려해 보안 오버헤드를 최소화하는 알고리즘 설계가 필요하다. Simulink를 통해 다양한 공격 시나리오(스푸핑, 리플레이)를 재현하고, 보안 알고리즘의 성능(지연 시간, 리소스 사용)을 평가한다.

2.2.5 학습 내용 정리

CAN 프로토콜 학습을 통해 메시지 프레임 구조, 우선순위 처리, 오류 검출 메커니즘을 이해했다. 보안 취약점으로는 인증 부족, 암호화 부재, DoS 공격 가능성을 도출했으며, 이를 해결하기 위해 메시지 인증, 데이터 암호화, 이상 탐지 기법을 검토했다. Simulink에서는 CAN 통신 블록을 활용해 ECU 간 통신을 구현하고, 보안 알고리즘 적용 후 네트워크 성능을 분석한다. 이는 부품별 보안 최적화와 다양한 알고리즘 비교를 위한 기초 데이터를 제공하며, 프로젝트의 핵심 목표인 연계 보안 기술 개발에 기여한다.

2.3 CAN-FD

2.3.1 CAN-FD란?

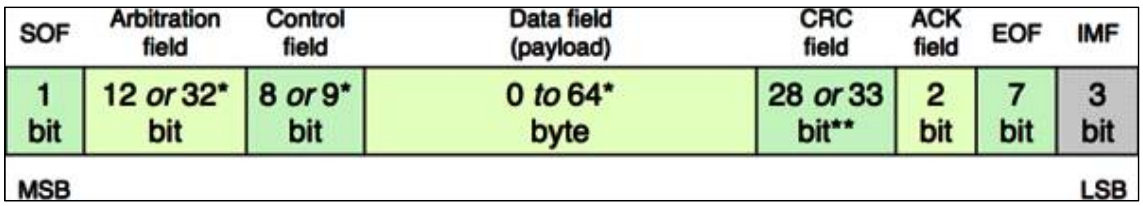


FIG4. CAN FD PROTOCOL

CAN-FD(Flexible Data Rate)는 기존 CAN의 한계를 보완하기 위해 Bosch에서 제안한 확장 규격으로, 본 프로젝트의 핵심 연구 대상 중 하나이다. CAN-FD는 기존 8바이트로 제한된 데이터 필드(Data Field)를 최대 64바이트까지 확장할 수 있으며, 데이터 구간에서 비트 레이트를 기존보다 높게 설정할 수 있어 대역폭 활용 효율과 전송 속도가 크게 향상된다. 이에 따라, 보안 알고리즘 적용 시 발생하는 오버헤드(추가 바이트, 계산 지연)를 수용할 수 있는 장점이 있다.

2.3.2 CAN-FD의 구조 및 특징

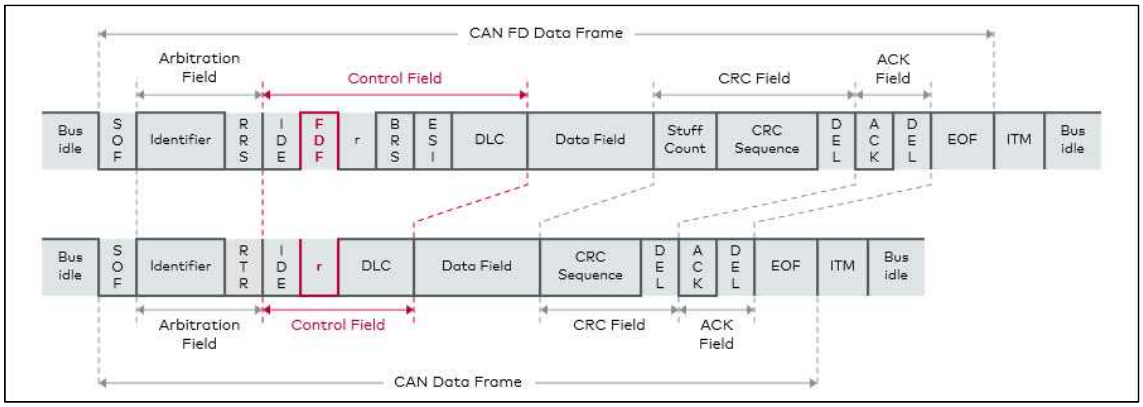


FIG5. CAN FD FRAME vs. CAN

CAN-FD의 프레임은 기존 CAN과 유사한 구조를 가지나, 몇 가지 주요 차별점이 존재한다.

(1) 데이터 필드 확장: 최대 64바이트 지원 → 보안 태그(MAC, HMAC, 암호화 데이터 등) 삽입 가능

(2) 비트 레이트 전환: 제어 구간(Control Field)과 데이터 구간(Data Field)에서 서로 다른 전송 속도를 사용 → 효율적 데이터 처리

(3) 호환성 유지: 기존 CAN 노드와 공존 가능, 단 CAN-FD 전용 기능은 지원되지 않음

이러한 특징은 보안 알고리즘을 ECU 간 통신에 적용할 때 유리하지만, 동시에 메시지 처리 지연이나 호환성 문제가 발생할 수 있다.

2.3.3 보안 취약점

CAN-FD는 데이터 용량과 속도를 확장했지만, 기본적인 보안 메커니즘은 여전히 제공하지 않는다.

(1) 메시지 위조: 더 큰 데이터 필드가 공격자에게 더 많은 위조 데이터 삽입 기회를 제공

(2) 리플레이 공격: 이전 프레임 재전송 공격은 그대로 유효

(3) DoS 공격: 대용량 데이터를 지속적으로 전송하여 네트워크 대역폭을 독점 가능

따라서 CAN-FD 역시 자체적으로는 보안이 취약하며, 외부 보안 알고리즘(예: MAC, 암호화) 적용이 필수적이다.

2.3.4 프로젝트와의 연계

본 프로젝트는 CAN-FD의 확장된 데이터 필드를 활용하여 **보안 알고리즘(HMAC-SHA256, Freshness Counter)**을 적용하는 데 중점을 둔다.

(1) 송신 ECU에서는 원시 데이터(AppData)와 Freshness Counter를 포함한 MAC을 계산하고, 이를 CAN-FD 프레임의 확장 데이터 필드에 삽입한다.

(2) 수신 ECU에서는 동일한 알고리즘으로 검증하여, 위·변조 또는 재전송 여부를 판별한다.

(3) Simulink Vehicle Network Toolbox를 활용해 CAN-FD Pack/Unpack 블록으로 ECU 간 메시지 흐름을 모델링하고, 보안 모듈을 삽입하여 보안성·지연 성능을 분석한다.

2.3.5 CAN vs. CAN FD

구분	CAN	CAN-FD
데이터 필드 길이	최대 8byte	최대 64byte
전송 속도	최대 1Mbps	최대 8Mbps
프레임 구조	Identifier(11/29bit), Data(0~8B), CRC, ACK	Identifier 동일, Data(0~64B), CRC 강화, Bit Rate Switch 필드 추가
보안 적용 용이성	데이터 필드 제한으로 MAC/암호화 삽입 어려움	확장된 데이터 필드로 암호화 블록 등 보안 데이터 삽입 가능
호환성	대부분 차량 ECU 적용	FD 지원 노드에서만 가능

2.4 TCP/IP 및 Ethernet 통신

2.4.1 TCP/IP 통신

TCP/IP 프로토콜은 인터넷 및 로컬 네트워크에서 데이터를 전송하기 위한 핵심 프로토콜 스택으로, 데이터가 네트워크를 통해 전달될 때의 구조와 절차를 정의한다. 일반적으로 4계층 구조(응용, 전송, 인터넷, 네트워크 접근 계층)로 설명되며, 이는 OSI 7계층 모델과 대응된다.

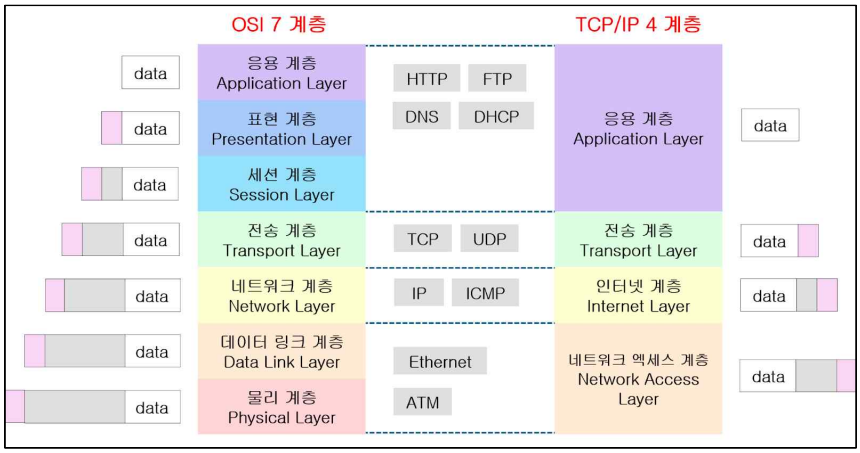


FIG6. OSI 및 TCP/IP 계층 구조

TCP/IP는 크게 두 가지 주요 구성 요소로 나뉜다:

(1) IP(Internet Protocol): 데이터 전송의 기반을 제공하는 비연결형, 비신뢰성 프로토콜로, 데이터를 패킷 단위로 나눠 전송한다. 전송 과정에서 순서 보장이나 오류 복구 기능이 없으며, 이러한 한계는 IP 주소 체계의 한계(특히 IPv4의 주소 고갈 문제)와 함께 차세대 프로토콜인 IPv6 도입의 필요성으로 이어지고 있다.

(2) TCP(Transmission Control Protocol): IP의 한계를 보완하여 신뢰성 있는 데이터 전송을 보장하는 연결형 프로토콜이다. 데이터의 정확한 전송과 순서를 보장하며, 전송 제어 및 오류 복구 기능을 포함하고 있다. TCP는 데이터 연결을 위해 3-Way Handshake 방식(SYN → SYN-ACK → ACK)을 통해 송신자와 수신자 간 연결을 설정한다.

이러한 TCP/IP 통신구조는 차량 내부 네트워크보다는 차량 외부 통신(예:V2X, OTA 업데이트)이나 고속 데이터 전송이 필요한 차량 내부 이더넷 구조에서 활용된다.

2.4.2 Ethernet 통신

이더넷(Ethernet)은 근거리(LAN) 및 광역(WAN) 통신망에서 가장 널리 사용되는 유선 통신 기술로, 자동차 분야에서도 고속 데이터 전송이 요구되는 센서 및 ECU 간 통신 수단으로 점차 확대되고 있다. 다음은 네트워크망의 구성도(FIG2)이다.

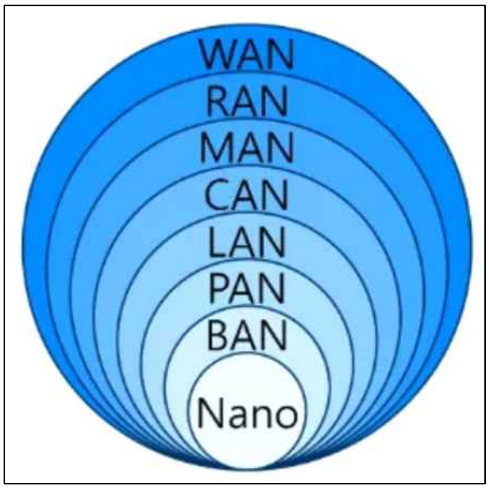


FIG7. 네트워크망의 구성도

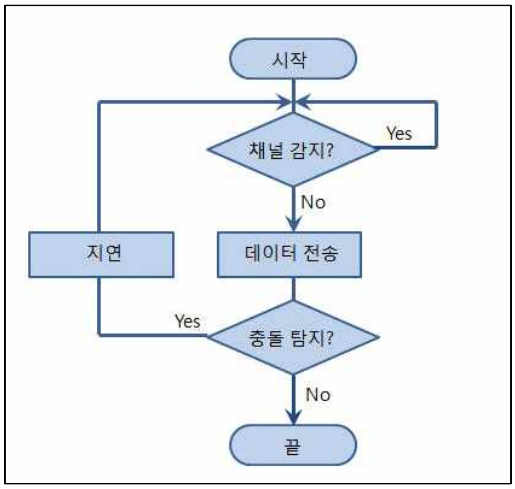


FIG8. CSMA/CD 순서도

이더넷은 OSI 모델의 물리 계층(L1)과 데이터 링크 계층(L2)에 해당하며, MAC(Media Access Control) 주소 기반의 고유 식별 체계를 통해 통신을 수행한다. 이더넷 통신에서는 CSMA/CD(FIG3, Carrier Sense Multiple Access with Collision Detection) 방식이 적용되어 다음과 같은 순서로 데이터 충돌을 감지하고 제어한다.

(1) Carrier Sensing: 전송 회선이 비어(IDLE) 있는지 확인

(2) 전송 중인 데이터가 없으면: 데이터 전송 시작

(3) 전송 도중 충돌 감지 시: 충돌 신호(Collision Signal) 발생

(4) Jam Signal 전송 후: 랜덤한 시간 대기 후 재시도

이더넷은 기본적으로 ACK 응답을 사용하지 않으며, 충돌 발생 시 자체적으로 충돌 감지 후 재전송을 수행하는 구조이다.

차량용 Ethernet 통신의 경우 one-pair UTP(Unshield Twisted Pair) 통신선을 사용하는 구조로 새롭게 설계되었다. 차량 내에 제어 데이터와 멀티미디어 데이터 처리를 위한 모든 요구 사항을 만족하고, 다양한 종류의 데이터를 실시간으로 전달하고 동기화하기 위해 별도의 통신 프로토콜을 개발하였다.

TCP/IP 및 이더넷 통신에 대한 학습은 자동차 보안 알고리즘 설계 시 각 프로토콜의 동작 원리, 한계, 그리고 보안 취약점을 고려하는 데 있어 필수적인 기반이 된다.

2.5 개발 환경 및 사용 기술

2.5.1 개발 환경

(1) MATLAB/Simulink

ㄱ. 자동차 전장부품(Brake ECU, Dashboard ECU 등) 모델링

ㄴ. CAN-FD 및 Ethernet 통신 시뮬레이션

ㄷ. Vehicle Network Toolbox활용 CAN-FD Pack/Unpack, UDP/TCP 전송 모델링

ㄹ. 보안 알고리즘(HMAC-SHA256, Freshness Counter) 삽입 및 검증 환경 구축

(2) VS Code

ㄱ. MATLAB 스크립트와 일부 보안 알고리즘 테스트 코드 관리

(3) MATLAB App Designer

ㄱ. UI 설계(보안 알고리즘 선택, 시뮬레이션 결과 시각화)

2.5.2 사용 언어

(1) MATLAB: Simulink 모델링, CAN-FD/Ethernet 통신, 보안 알고리즘 구현 및 결과 분석

(2) Python 3.10: 보조적 알고리즘 검증, 데이터 분석, 시각화

2.5.3 사용 기술

(1) 통신 및 보안

ㄱ. Vehicle Network Toolbox: CAN-FD 프레임 생성, ECU 간 통신 모델링

ㄴ. Instrument Control Toolbox: UDP/TCP 블록 활용 → Ethernet 통신 경로 구현

ㄷ. 보안 알고리즘: HMAC-SHA256 기반 무결성 검증, Freshness Counter 기반 리플레이 공격 방어

(2) UI 및 시각화

ㄱ. MATLAB App Designer: 알고리즘 선택, 시뮬레이션 결과(무결성, Freshness, 지연) 시각화

ㄴ. Scope/Display 블록: ECU 출력값과 보안 검증 결과 모니터링

(3) 배포 및 환경 관리

ㄱ. Docker(선택적): MATLAB Runtime 기반 컨테이너 환경에서 Simulink 실행 가능성 검토

ㄴ. GitHub: Simulink 모델 및 코드 관리, 버전 관리

3. 연구 내용

3.1 프로젝트 구성

본 프로젝트는 MATLAB/Simulink 환경에서 송신부(TX)와 수신부(RX)로 구성된 차량 통신 보안 시스템을 구현하는 것을 목표로 한다. 기본 통신 경로는 CAN-FD(Controller Area Network – Flexible Data rate)를 기반으로 하며, 여기에 Ethernet(UDP 기반)을 병렬로 확장하여 다중 프로토콜 환경에서의 보안성을 검증하였다.

(1) CAN-FD 경로: 브레이크 ECU 데이터를 24바이트 프레임으로 구성하여 전송, 수신 ECU에서 MAC/Freshness 검증 수행

(2) Ethernet 경로: 동일 데이터를 18바이트 PDU로 포맷하여 UDP로 송수신, 수신부에서 MAC/Freshness 검증 후 ECU 로직에 반영

이를 통해 CAN-FD의 실시간성·호환성과 Ethernet의 고속·대용량성을 동시에 반영한 하이브리드 보안 통신 모델을 구축하였다.

3.2 송신부 (TX) 구현

송신부는 브레이크 ECU(BrakeSensorECU)의 역할을 모사하며, **센서 신호 생성 → 데이터 패킹 → MAC 생성 → 최종 페이로드 조립 → CAN-FD 전송 & Ethernet 전송** 순서로 구성되었다.

(1) 센서 신호 생성

ㄱ. Brake_Pedal_Raw: Slider 블록을 통해 사용자가 직접 값을 제어하도록 설계

ㄴ. Brake_Pressed: Threshold 비교 후 0/1 신호 출력

ㄷ. Freshness Counter: Counter Free-Running 블록으로 매주기 증가하는

카운터 생성

(2) 데이터 패킹

ㄱ. Byte Pack (BE) 블록을 통해 Raw(2B) + Pressed(1B)를 3바이트 (AppData)로 변환

ㄴ. Freshness Counter(uint32)를 별도로 4바이트로 유지

(3) MAC 생성

ㄱ. MATLAB Function 블록에서 **HMAC-SHA256** 구현

ㄴ. 입력: AppData(3B) + Freshness(4B), 출력: 32B MAC

ㄷ. 상위 8바이트만 추출 → MAC8

(4) 최종 페이로드 조립 (CAN-FD)

ㄱ. Vector Concatenate: AppData(3B) + Freshness(4B) + MAC8(8B) + Padding(9B) = 총 24B

ㄴ. CAN FD Pack/Transmit 블록 → Virtual CAN 채널로 전송

(5) Ethernet용 병렬 출력

ㄱ. 송신부 내부에서 동시에 **Outport 4개** 구성 (Out_raw, Out_pressed, Out_freshness, Out_mac8)

ㄴ. 이 신호들을 **ETH_TX Subsystem**으로 전달하여 UDP 전송에 활용

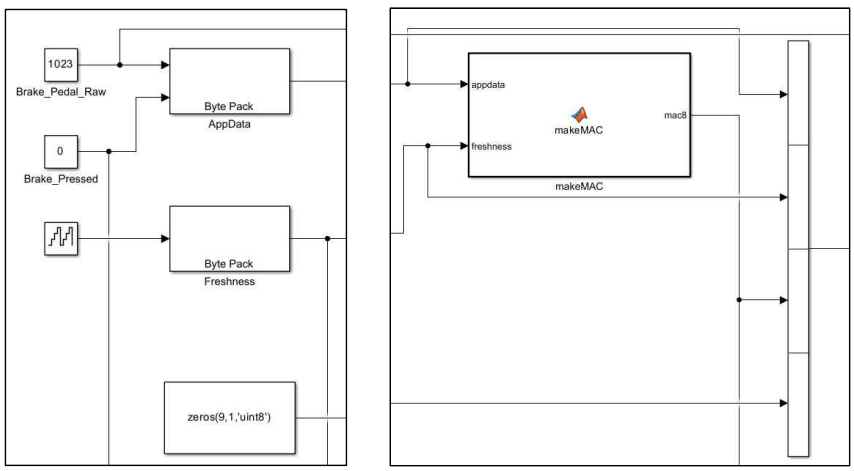


FIG9, 10. 송신부(TX) 1

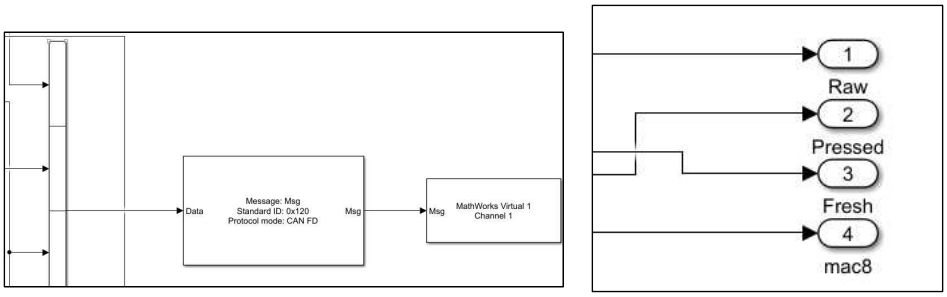


FIG11, 12. 송신부(TX) 2

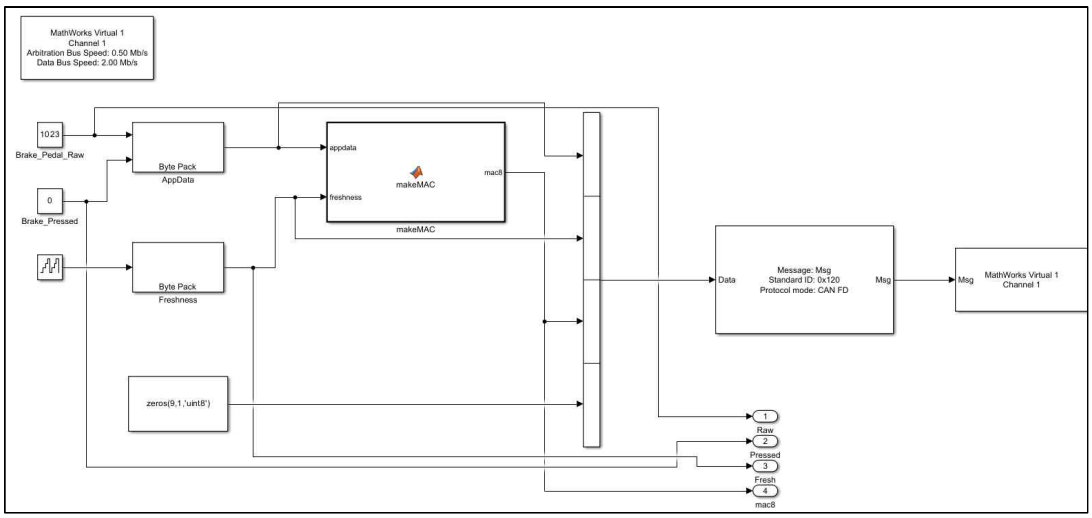


FIG13. 송신부(TX) FLOW CHART

3.3 수신부 (RX) 구현

수신부는 ABSECU의 역할을 수행하며, **CAN-FD 수신 경로**와 **Ethernet 수신 경로**를 병렬로 배치하였다.

(1) CAN-FD 수신 경로

ㄱ. CAN FD Receive 블록 → 24B 프레임 수신

ㄴ. Byte Unpack으로 AppData, Freshness, MAC8 분리

ㄷ. makeMAC 함수 재실행 → 수신 MAC8과 비교 (mac_ok)

ㄹ. Freshness 검증 함수 → fv_ok

ㅁ. valid = mac_ok AND fv_ok

ㅂ. Switch 블록: valid=1일 때만 Brake_Pedal_Raw_rx, Brake_Pressed_rx를

ECU 로직으로 전달

(2) Ethernet 수신 경로 (ETH_RX)

ㄱ. UDP Receive(Local Port=15000) → uint8[1×18] PDU 수신

ㄴ. Signal Specification으로 크기/타입 고정

ㄷ. MATLAB Function(parsePDU) → ID, DLC, Raw, Pressed, Freshness, MAC8 추출

ㄹ. MAC 검증: ID+DLC+AppData+Freshness 기반 바이트열 재구성 후 HMAC-SHA256 계산 → 수신 MAC8과 비교

ㅁ. Freshness 검증: 수신 Freshness vs 이전 값 비교 → 재전송 여부 판정

ㅂ. 최종 valid_eth 생성 → Switch 통해 ECU 로직 연결

(3) 최종 ECU 로직

ㄱ. CAN vs Ethernet 입력은 Variant Source/Switch로 선택 가능

ㄴ. Gauge, Lamp 블록에 연결하여 인증된 데이터일 때만 계기 및 경고등 작동

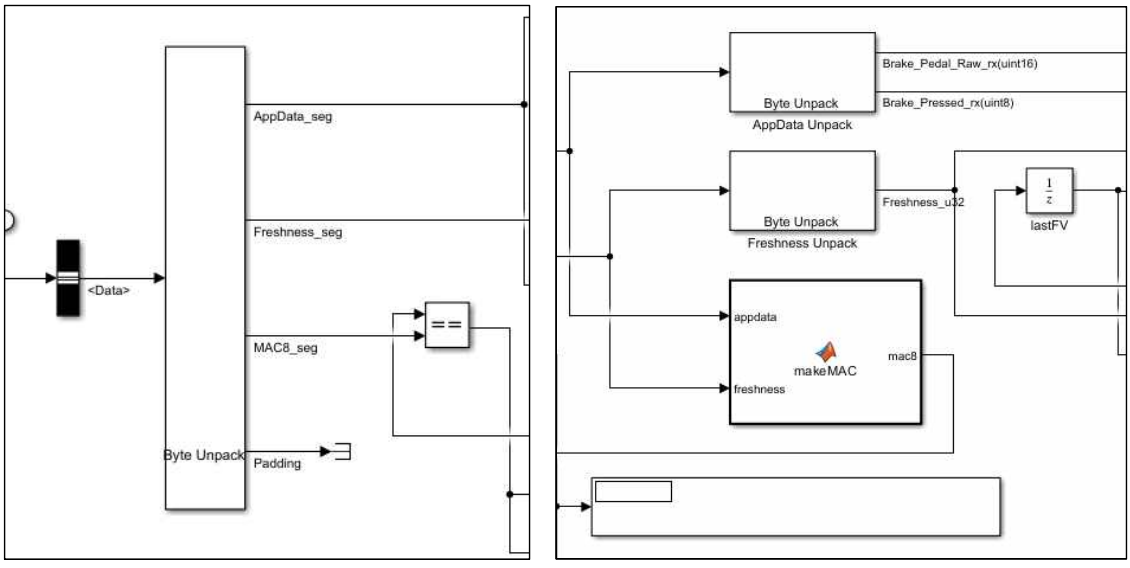


FIG14, 15. 수신부(RX) 1

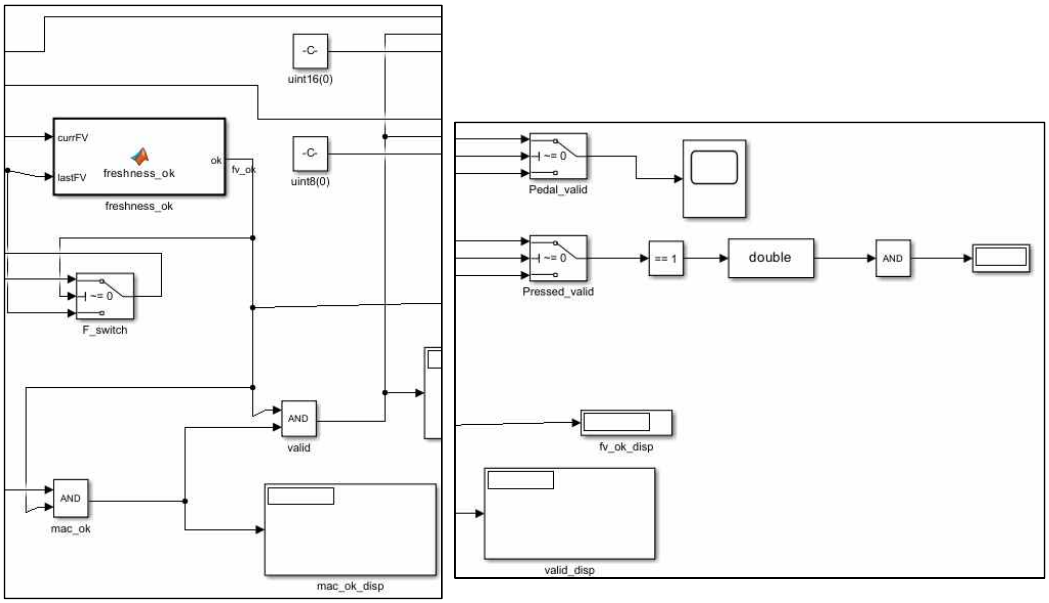


FIG16, 17. 수신부(RX) 2

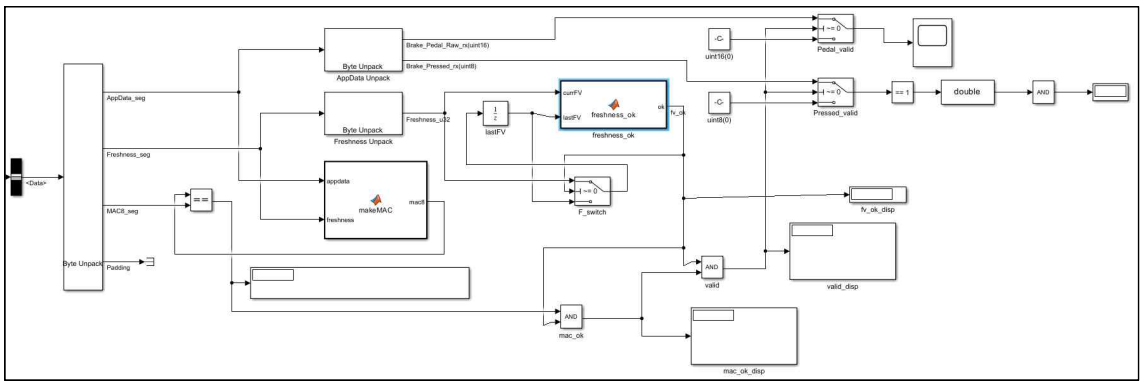


FIG18. 수신부(RX) FLOW CHART

3.4 Ethernet 송신부 (ETH_TX) 구현

ETH_TX는 BrakeSensorECU_TX의 Outport 신호(Raw, Pressed, Freshness, MAC8)를 받아 UDP 전송용 PDU로 조립한다.

(1) AppData 구성: Raw(uint16) → Byte Pack(BE, 2B), Pressed(uint8, 1B) → Vector Concatenate(3B)

(2) Freshness 변환: Freshness(uint32) → Byte Pack(BE, 4B)

(3) ID/DLC 생성: Constant 블록에서 ID=0x0200, DLC=18 지정 → Shift Arithmetic/Bitwise로 id_hi, id_lo 분리

(4) PDU 조립: [ID(2B), DLC(1B), AppData(3B), Freshness(4B), MAC8(8B)] = 18B

(5) 출력 고정: Signal Specification (uint8[1×18])

(6) 전송: UDP Send(Remote IP, Port=15000)

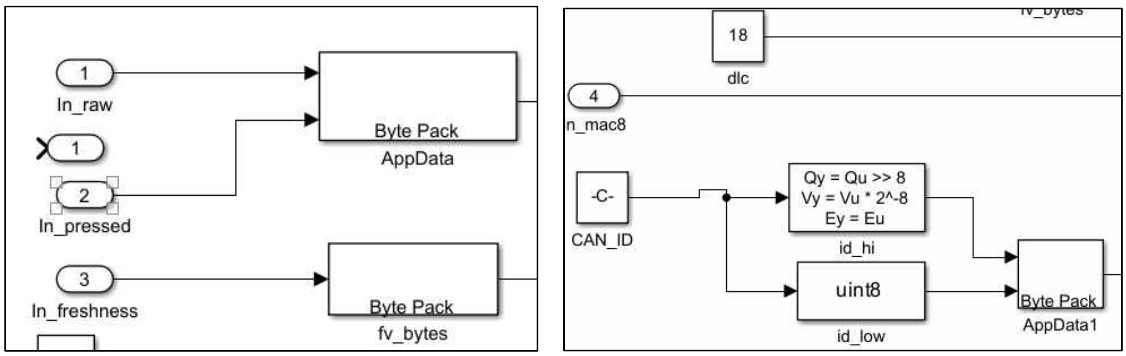


FIG19, 20. ETH_TX

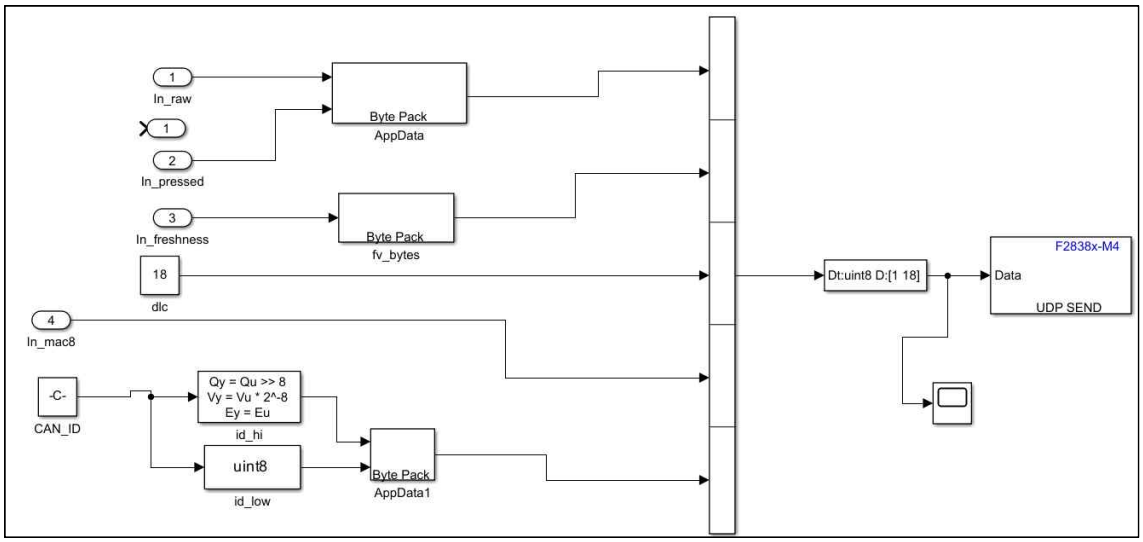


FIG21. ETH_TX FLOW CHART

3.5 Ethernet 수신부 (ETH_RX) 구현 (예정)

ETH_RX는 UDP Receive 블록을 시작점으로 하여 PDU를 파싱, 검증, ECU 로직에 연결한다.

(1) UDP Receive: Local Port=15000, 출력 uint8[1×18]

(2) PDU 파싱: MATLAB Function(parsePDU)

- ID(2B), DLC(1B), Raw(2B), Pressed(1B), Freshness(4B), MAC8(8B) 추출

(3) MAC 검증: 송신부와 동일한 방식으로 HMAC-SHA256 재계산 → 수신 MAC8과 비교

(4) Freshness 검증: Unit Delay로 lastFV 저장 → (currFV>lastFV)&&(currFV-lastFV≤W) 조건 검사

(5) 유효성 판정: mac_ok AND fv_ok = valid_eth

(6) 데이터 게이트: Switch 블록 → valid_eth=1일 때만 Raw, Pressed 전달

(7) ECU 로직 반영: Gauge, Lamp 블록 연결

3.6 HMAC-SHA256 기반 MAC 생성 함수 구현

3.6.1 목적과 요약

본 절에서는 송신부(TX)에서 사용한 **메시지 인증 코드(MAC)** 생성 함수의 내부 구조와 동작을 기술한다. 함수는 **AppData(3B)**와 **Freshness(4B)**를 연결한 7바이트 메시지에 대해 **HMAC-SHA256**을 계산하고, 결과 32바이트 중 **상위 8바이트(MAC8)**만 반환하여 CAN-FD 페이로드에 포함한다. 설계 핵심은 **고정 크기 버퍼**와 **정해진 루프 경계**를 사용해 **Simulink 코드 생성** 친화적으로 만든 점이다.

3.6.2 인터페이스(입·출력, 자료형)

ㄱ. 함수: mac8 = makeMAC(appdata, freshness)

ㄴ. 입력

appdata: uint8[3x1] (Raw 2B + Pressed 1B, Big-Endian)

freshness: uint8[4x1] (Freshness Counter, Big-Endian)

ㄷ. 출력

mac8: uint8[8x1] (HMAC-SHA256 결과 상위 8바이트, 트렁케이션)

- 주의: 블록 내부에서 appdata(:)/freshness(:)로 열벡터를 강제하여 차원 오류를 방지한다.

3.6.3 구성(상위→하위 함수)

(1) makeMAC

ㄱ. 키(16B, 예제용) 정의 → 메시지(msg = [appdata; freshness]) 구성 → hmac_sha256_fixed 호출 → 32B 결과의 상위 8B 반환.

(2) hmac_sha256_fixed

ㄱ. HMAC 표준 절차 수행.

ㄴ. **64B 블록 크기** 기준으로 키 정규화(>64B면 SHA256(key)로 32B 압축, 나머지 0패딩).

ㄷ. ipad = key ⊕ 0x36, opad = key ⊕ 0x5c (각 64B).

ㄹ. **inner** = SHA256(ipad || msg) (길이 64+7=71B)

ㅁ. **outer** = SHA256(opad || inner) (길이 64+32=96B).

ㅂ. 두 경우 모두 패딩 후 **정확히 128바이트(2블록)** 처리로 고정.

(3) sha256_bytes_fixed_concat(part1,len1, part2,len2)

ㄱ. part1||part2를 **128B 고정 버퍼 BUF**에 배치하고, 0x80 패딩 및 메시지 길이(비트)를 마지막 8B(빅엔디언)에 기록.

ㄴ. 64B씩 두 블록 순차 압축(SHA-256 라운드 64회×2).

ㄷ. 결과 8×uint32 상태값을 **빅엔디언 바이트 32개**로 직렬화.

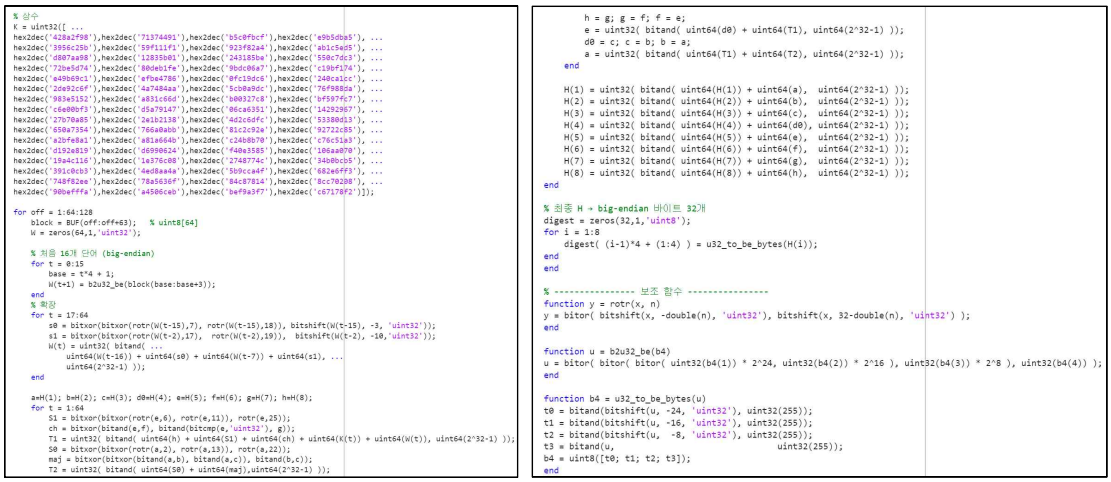
(4) 보조 함수

ㄱ. rotr(우회전), b2u32_be(4B→uint32), u32_to_be_bytes(uint32→4B BE).

```
function mac8 = makeMAC(appdata, freshness)
%#codegen
% appdata  : uint8[3x1]
% freshness: uint8[4x1]
% mac8     : uint8[8x1] (HMAC-SHA256의 앞 8바이트)

% --- 1) 키 (송/수신 동일) ---
key = uint8([1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16].'; % [16x1]

% --- 2) 메시지: appdata(3) || freshness(4) = 7바이트 ---
msg = uint8([appdata(:); freshness(:)]);  % [7x1]

% --- 3) HMAC-SHA256 (고정 길이 전용) ---
fullmac = hmac_sha256_fixed(key, msg);    % uint8[32x1]

% --- 4) 상위 8바이트만 사용 ---
mac8 = fullmac(1:8);
end

% ============================================================
% HMAC-SHA256 (고정 길이 전용: ipad/opad 64B + (7B 또는 32B))
% ============================================================
function out = hmac_sha256_fixed(key, msg)
%#codegen
% key : uint8[<=64x1] (여기선 16B)
% msg : uint8[7x1]

blockSize = uint32(64);

% 1) 키를 64바이트로 정규화(>64면 SHA256(key)로 32바이트, 나머지 0패딩)
fixedKey = zeros(blockSize,1,'uint8');     % [64x1]
k = key(:);
lenK = uint32(numel(k));
if lenK > blockSize
    k = sha256_bytes_fixed_concat(zeros(0,1,'uint8'), 0, k, numel(k)); % 일반 SHA256(key)
    lenK = uint32(numel(k));  % = 32
end
fixedKey(1:double(lenK)) = k; % 앞부분만 채우고 나머지 0

% 2) ipad/opad (64바이트)
ipad = bitxor(fixedKey, repmat(uint8(54), 64, 1)); % 0x36
opad = bitxor(fixedKey, repmat(uint8(92), 64, 1)); % 0x5C

% 3) inner = SHA256( ipad || msg )  → (64 + 7)B = 71B → 패딩 후 항상 128B
inner = sha256_bytes_fixed_concat(ipad, 64, msg, 7);  % uint8[32x1]
```

```
% ============================================================
% SHA256( part1[64] || part2[N] ) 전용: 전체 길이가 71B 또는 96B인 경우
% → 패딩 후 항상 128바이트(2블록) 고정 버퍼로 처리(동적크기 없음)
% ============================================================
function digest = sha256_bytes_fixed_concat(part1, len1, part2, len2)
%#codegen
% part1: uint8[64x1] 또는 빈 벡터(일반 SHA용)
% len1 : 0 또는 64
% part2: uint8[Nx1]
% len2 : N (inner=7, outer=32, 일반 SHA에서는 임의)
% digest: uint8[32x1]

digest = zeros(32,1,'uint8');

% ----- 메시지 총 길이(바이트) -----
L = uint32(len1 + len2);

% ===== 특수 케이스: 일반 SHA256(키 압축 등) - 길이가 56 미만일 수도 있음
% 여기서는 최대 64바이트 키를 처리: 길이 <= 64 → 패딩 후 한 블록 또는 두 블록
% 하지만 간단화를 위해 항상 두 블록(128바이트) 버퍼로 처리해도 안전
BUF = zeros(128,1,'uint8');   % 고정 크기

% 1) 원 데이터 복사
if len1 > 0
    BUF(1:len1) = part1(1:len1);
end
if len2 > 0
    BUF(len1+1 : len1+len2) = part2(1:len2);
end

% 2) 0x80 추가
pos = double(L) + 1;
BUF(pos) = uint8(128);

% 3) 길이(비트) = L * 8 → 마지막 8바이트(big-endian) 기록
mlen_bits = uint64(L) * 8;
% big-endian으로 BUF(121:128)에 저장 (128-8+1=121)
for i = 0:7
    BUF(128 - i) = uint8(bitand(bitshift(mlen_bits, -8*i, 'uint64'), uint64(255)));
end
```

FIG22, 23. make MAC 1



```
% 상수
K = uint32([ ...
hex2dec('428a2f98'),hex2dec('71374491'),hex2dec('b5c0fbcf'),hex2dec('e9b5dba5'), ...
hex2dec('3956c25b'),hex2dec('59f111f1'),hex2dec('923f82a4'),hex2dec('ab1c5ed5'), ...
hex2dec('d807aa98'),hex2dec('12835b01'),hex2dec('243185be'),hex2dec('550c7dc3'), ...
hex2dec('72be5d74'),hex2dec('80deb1fe'),hex2dec('9bdc06a7'),hex2dec('c19bf174'), ...
hex2dec('e49b69c1'),hex2dec('efbe4786'),hex2dec('0fc19dc6'),hex2dec('240ca1cc'), ...
hex2dec('2de92c6f'),hex2dec('4a7484aa'),hex2dec('5cb0a9dc'),hex2dec('76f988da'), ...
hex2dec('983e5152'),hex2dec('a831c66d'),hex2dec('b00327c8'),hex2dec('bf597fc7'), ...
hex2dec('c6e00bf3'),hex2dec('d5a79147'),hex2dec('06ca6351'),hex2dec('14292967'), ...
hex2dec('27b70a85'),hex2dec('2e1b2138'),hex2dec('4d2c6dfc'),hex2dec('53380d13'), ...
hex2dec('650a7354'),hex2dec('766a0abb'),hex2dec('81c2c92e'),hex2dec('92722c85'), ...
hex2dec('a2bfe8a1'),hex2dec('a81a664b'),hex2dec('c24b8b70'),hex2dec('c76c51a3'), ...
hex2dec('d192e819'),hex2dec('d6990624'),hex2dec('f40e3585'),hex2dec('106aa070'), ...
hex2dec('19a4c116'),hex2dec('1e376c08'),hex2dec('2748774c'),hex2dec('34b0bcb5'), ...
hex2dec('391c0cb3'),hex2dec('4ed8aa4a'),hex2dec('5b9cca4f'),hex2dec('682e6ff3'), ...
hex2dec('748f82ee'),hex2dec('78a5636f'),hex2dec('84c87814'),hex2dec('8cc70208'), ...
hex2dec('90befffa'),hex2dec('a4506ceb'),hex2dec('bef9a3f7'),hex2dec('c67178f2')]);

for off = 1:64:128
    block = BUF(off:off+63);   % uint8[64]
    W = zeros(64,1,'uint32');

    % 처음 16개 단어 (big-endian)
    for t = 0:15
        base = t*4 + 1;
        W(t+1) = b2u32_be(block(base:base+3));
    end
    % 확장
    for t = 17:64
        s0 = bitxor(bitxor(rotr(W(t-15),7), rotr(W(t-15),18)), bitshift(W(t-15), -3, 'uint32'));
        s1 = bitxor(bitxor(rotr(W(t-2),17),  rotr(W(t-2),19)),  bitshift(W(t-2), -10,'uint32'));
        W(t) = uint32( bitand( ...
            uint64(W(t-16)) + uint64(s0) + uint64(W(t-7)) + uint64(s1), ...
            uint64(2^32-1) ));
    end

    a=H(1); b=H(2); c=H(3); d=H(4); e=H(5); f=H(6); g=H(7); h=H(8);
    for t = 1:64
        S1 = bitxor(bitxor(rotr(e,6), rotr(e,11)), rotr(e,25));
        ch = bitxor(bitand(e,f), bitand(bitcmp(e,'uint32'), g));
        T1 = uint32( bitand( uint64(h) + uint64(S1) + uint64(ch) + uint64(K(t)) + uint64(W(t)), uint64(2^32-1) ));
        S0 = bitxor(bitxor(rotr(a,2), rotr(a,13)), rotr(a,22));
        maj = bitxor(bitxor(bitand(a,b), bitand(a,c)), bitand(b,c));
        T2 = uint32( bitand( uint64(S0) + uint64(maj),uint64(2^32-1) ));
        h = g; g = f; f = e;
        e = uint32( bitand( uint64(d0) + uint64(T1), uint64(2^32-1) ));
        d0 = c; c = b; b = a;
        a = uint32( bitand( uint64(T1) + uint64(T2), uint64(2^32-1) ));
    end

    H(1) = uint32( bitand( uint64(H(1)) + uint64(a),  uint64(2^32-1) ));
    H(2) = uint32( bitand( uint64(H(2)) + uint64(b),  uint64(2^32-1) ));
    H(3) = uint32( bitand( uint64(H(3)) + uint64(c),  uint64(2^32-1) ));
    H(4) = uint32( bitand( uint64(H(4)) + uint64(d0), uint64(2^32-1) ));
    H(5) = uint32( bitand( uint64(H(5)) + uint64(e),  uint64(2^32-1) ));
    H(6) = uint32( bitand( uint64(H(6)) + uint64(f),  uint64(2^32-1) ));
    H(7) = uint32( bitand( uint64(H(7)) + uint64(g),  uint64(2^32-1) ));
    H(8) = uint32( bitand( uint64(H(8)) + uint64(h),  uint64(2^32-1) ));
end

% 최종 H → big-endian 바이트 32개
digest = zeros(32,1,'uint8');
for i = 1:8
    digest( (i-1)*4 + (1:4) ) = u32_to_be_bytes(H(i));
end
end

% ---------- 보조 함수 ----------
function y = rotr(x, n)
y = bitor( bitshift(x, -double(n), 'uint32'), bitshift(x, 32-double(n), 'uint32') );
end

function u = b2u32_be(b4)
u = bitor( bitor( bitor( uint32(b4(1)) * 2^24, uint32(b4(2)) * 2^16 ), uint32(b4(3)) * 2^8 ), uint32(b4(4)) );
end

function b4 = u32_to_be_bytes(u)
t0 = bitand(bitshift(u, -24, 'uint32'), uint32(255));
t1 = bitand(bitshift(u, -16, 'uint32'), uint32(255));
t2 = bitand(bitshift(u,  -8, 'uint32'), uint32(255));
t3 = bitand(u,                          uint32(255));
b4 = uint8([t0; t1; t2; t3]);
end
```

FIG24, 25. make MAC 2

3.6.4 동작 상세(핵심 로직)

(1) 키 정규화: HMAC 규격에 따라 키를 64B로 맞춘다(>64B면 SHA-256으로 32B 압축 후 0패딩, ≤64B면 앞부분에 복사).

(2) ipad/opad 적용: 각각 64B 배열과 XOR하여 내부/외부 해시의 시드를 만든다.

(3) 패딩 처리: BUF(128B)에 원데이터를 넣고 0x80 바이트 추가, 메시지 총길이(bit)를 마지막 8바이트에 BE로 기록. 길이가 71B(inner)·96B(outer)인 특수 경우로 항상 2블록을 사용하므로 동적 크기/가변 루프가 없다.

(4) 라운드 계산: 메시지 스케줄 W[64] 확장(σ0, σ1), 상태 a~h 갱신(Σ0, Σ1, Ch, Maj). 덧셈은 uint64로 올려 연산 후 2^32-1 마스킹으로 uint32 오버플로를 제어 → 정확한 모듈러 2^{32} 구현.

3.6.5 보안적 고려사항

(1) 트렁케이션(8바이트): 32B 중 8B만 사용 → 대역폭 절감 장점 vs. 안전성 저하 가능성. 위협 모델에 따라 12~16B로 확대 고려.

(2) 키 관리: 현재 코드는 하드코딩 키(예제). 실제 시스템에서는 **Mask 파라미터/보안 저장소**를 통해 주입, 키 롤오버 전략 필요.

(3) 메시지 범위: 본 함수는 **CAN-FD** 경로의 AppData||Freshness 7B만 서명.

ㄱ. Ethernet 경로(본 과제의 PDU 규격)에서는 ID||DLC||AppData||Freshness(10B)를 서명 대상으로 권장. 이 경우 래퍼 함수(makeMAC_eth)로 10B 메시지를 구성해 동일 HMAC 코어에 전달하도록 확장 가능(아래 2.8.8 참조).

(4) 동기화: Freshness 오차 허용 윈도우(W) 정의/관리 필요. 시뮬레이터와 ECU 간 주기 차이 고려.

3.6.6 요약

본 프로젝트의 MAC 생성 모듈은 고정 길이 HMAC-SHA256을 코드 생성 친화적 구조로 구현하여, AppData||Freshness(7B) 메시지에 대한 무결성 태그를 생성하고 8바이트로 트렁케이션한다. 128B 고정 버퍼·정적 루프·빅엔디언 직렬화를 통해 Simulink/Embedded Coder 환경에서 안정적인 실시간 동작을 보장하며, Ethernet PDU(10B 서명 입력)에도 소규모 확장으로 동일 코어를 적용 가능하다.

3.7 Freshness Counter 검증 함수 설계

본 절에서는 재전송 공격(Replay Attack)을 방지하기 위한 **Freshness Counter** 검증 로직을 설명한다. Freshness Counter는 송신 ECU가 매 주기마다 증가시키는 고유 값으로, 수신 ECU는 이를 검증하여 과거 메시지의 재전송 여부를 판별한다. 본 프로젝트에서는 MATLAB Function 블록을 통해 다음과 같이 구현하였다.

```
function ok = freshness_ok(currFV, lastFV)
%#codegen
W = uint32(1024);   % 허용 윈도우
ok = (currFV >= lastFV) && (currFV - lastFV <= W);
end
```

FIG26. Freshness Counter

3.7.1 함수 구조와 동작 원리

(1) 입력 값

currFV: 현재 수신된 Freshness Value

lastFV: 이전에 저장된 Freshness Value

(2) 출력 값

ok: Boolean 값. 조건을 만족하면 true, 그렇지 않으면 false

함수는 크게 두 가지 조건을 동시에 만족해야 참(true)을 반환한다.

(3) currFV >= lastFV: Freshness는 반드시 단조 증가해야 하므로, 수신 값이 이전 값보다 작으면 공격으로 간주한다.

(4) (currFV - lastFV <= W): 증가 폭이 허용 윈도우(W=1024)를 초과하지 않아야 한다. ECU 간 동기화 오차나 네트워크 지연으로 인해 일부 값 건너뜀이 발생할 수 있으므로, 이를 허용하기 위한 윈도우를 설정하였다.

3.7.2 보안적 의의

(1) 재전송 공격 방지

공격자가 과거의 정상 메시지를 그대로 재전송하는 경우, currFV < lastFV 조건에서 탐지된다.

(2) 동기화 허용

ECU 간 주기 차이나 지연으로 인해 Freshness가 다소 불규칙하게 증가할 수 있다. 이때 윈도우 W를 설정함으로써 정상 메시지를 불필요하게 거부하지 않도록 한다.

(3) 연산 효율성

Q단순한 비교와 뺄셈 연산으로 구성되어 있어, 실시간성을 요구하는 차량 통신 환경에 적합하다.

3.8 보안 검증 결과 시각화 UI 설계

본 절에서는 MAC 및 Freshness 검증 결과를 **직관적으로 확인하기 위한 UI(User Interface) 구성**을 설명한다. 본 프로젝트는 단순한 시뮬레이션을 넘어, 실제 운용 환경에서 엔지니어가 보안 상태를 빠르게 인식할 수 있도록 시각화 기능을 추가하였다.

3.8.1 UI 구성 요소

UI는 Simulink Dashboard 컴포넌트와 App Designer 기반 위젯을 활용하여 다음과 같이 구성하였다.

(1) Brake Pedal Raw Value (슬라이더)

ㄱ. 0~1023 범위의 아날로그 값으로, 브레이크 페달 위치를 모의 입력.

ㄴ. 사용자가 슬라이더를 조작함으로써 Brake_Pedal_Raw 데이터가 변동한다.

(2) Brake Pressed 여부 (스위치)

ㄱ. On/Off 스위치 형태로 페달 눌림 상태를 이진 신호로 출력.

ㄴ. 정상 입력일 경우 초록색 체크 표시가 나타나 직관적으로 상태를 확인할 수 있다.

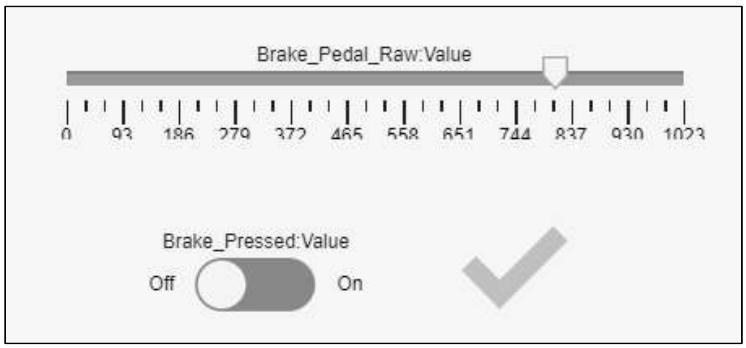


FIG27. Brake Pedal Raw Value / Brake Pressed

(3) 보안 위협 시뮬레이션 스위치

ㄱ. MAC 위변조 스위치: On으로 설정 시, MAC 검증 실패 상황을 강제로 발생시킨다.

ㄴ. Freshness 위변조 스위치: On으로 설정 시, Freshness 검증 실패(재전송 공격 등) 상황을 모의한다.

FIG28. MAC / Freshness 위변조 스위치

(4) 수신부 데이터 표시

ㄱ. 수신부의 Brake Pedal Raw Value와 Brake Pressed 여부를 텍스트 및 게이지로 표시.

ㄴ. 검증 성공 시 정상적으로 값이 갱신되며, 실패 시 기본값(0)으로 표시된다.

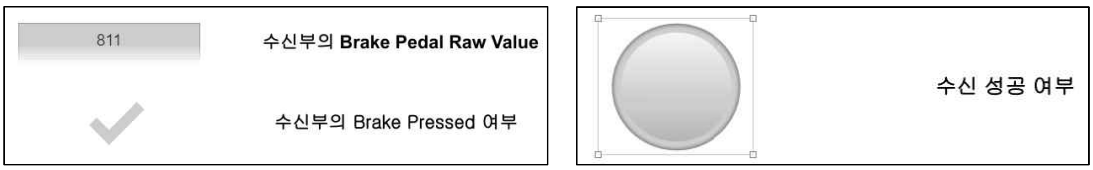


FIG29, 30. 수신부 데이터 / 수신 성공 여부

(5) 검증 결과 표시

ㄱ. MAC 불일치 시: 빨간색 X 표시

ㄴ. Freshness 오류 시: 회색 X 표시

ㄷ. 수신 성공 시: 초록색 체크와 원형 인디케이터가 점등

FIG31. 검증 결과 표시

3.8.2 동작 시나리오

(1) 정상 동작

ㄱ. 슬라이더와 스위치로 입력된 Raw 및 Pressed 값이 송신부를 통해 전송된다.

ㄴ. 수신부에서 MAC과 Freshness 검증을 모두 통과하면, UI의 수신 성공 여부가 체크되고, 램프/게이지에 정상 값이 반영된다.

(2) MAC 위변조 공격

ㄱ. 송신부에서 생성된 MAC8을 변조하거나, 수신부에서 강제로 위변조 스위치를 On으로 하면, 수신부 검증 단계에서 mac_ok=false가 된다.

ㄴ. 이때 UI는 MAC 불일치 빨간 X를 표시하며, Raw/Pressed 데이터는 ECU 로직으로 전달되지 않는다.

(3) Freshness 오류 (재전송 공격 등)

ㄱ. 동일한 Freshness 값으로 메시지가 재전송되거나, 허용 윈도우(W=1024)를 초과하는 값이 수신되면 fv_ok=false가 된다.

ㄴ. UI는 Freshness 오류 회색 X를 표시하며, 데이터는 차단된다.

3.8.3 보안적 의의

(1) 실시간 모니터링: 보안 검증 결과를 수치 값 대신 직관적인 시각 요소(체크, X, 램프)로 제공하여, 엔지니어가 즉시 인지할 수 있다.

(2) 공격 시뮬레이션: MAC 위변조와 Freshness 오류를 직접 On/Off 스위치로 제어함으로써, 다양한 공격 시나리오를 실험적으로 재현할 수 있다.

(3) Fail-Safe 검증: MAC 또는 Freshness 중 하나라도 실패하면 데이터가 ECU에 전달되지 않고 기본값으로 차단되는 구조를 시각적으로 확인할 수 있다.

4. 연구 결과 분석 및 평가

4.1. 시뮬레이션 방법

본 연구의 시뮬레이션은 MATLAB/Simulink 환경에서 송신부(TX)와 수신부(RX)를 구성하여 진행되었다. 송신부의 Slider 및 Switch 블록을 통해 브레이크 페달의 원시 데이터(Brake_Pedal_Raw)와 눌림 여부(Brake_Pressed)를 조작하면서, 다양한 정상·비정상 시나리오에 대해 수신부의 반응을 관찰하였다.

(1) 정상 동작:

Pedal_Raw=500, Pressed=1과 같은 정상 입력을 전송하면, 수신부에서 MAC 및 Freshness 검증이 모두 통과하여 Lamp가 점등됨을 확인하였다.

(2) Freshness 오류 시나리오:

동일한 Freshness 값이 반복 전송되면, freshness_ok 함수가 거짓(false)을 반환하여 Lamp는 꺼지도록 동작하였다.

(3) MAC 위변조 시나리오:

수신된 데이터나 MAC 값을 강제로 변경하면 mac_ok=false가 되어 Lamp가 꺼지며, 위변조 탐지가 가능함을 확인하였다.

(4) Pressed=0 시나리오:

ECU 로직 상에서 브레이크가 눌리지 않았다고 판정될 경우, valid 신호가 거짓으로 처리되어 제어 명령이 ECU로 전달되지 않았다.

이러한 시뮬레이션을 통해 CAN-FD 통신 환경에서의 데이터 무결성·신선도 검증 동작을 체계적으로 확인하였다.

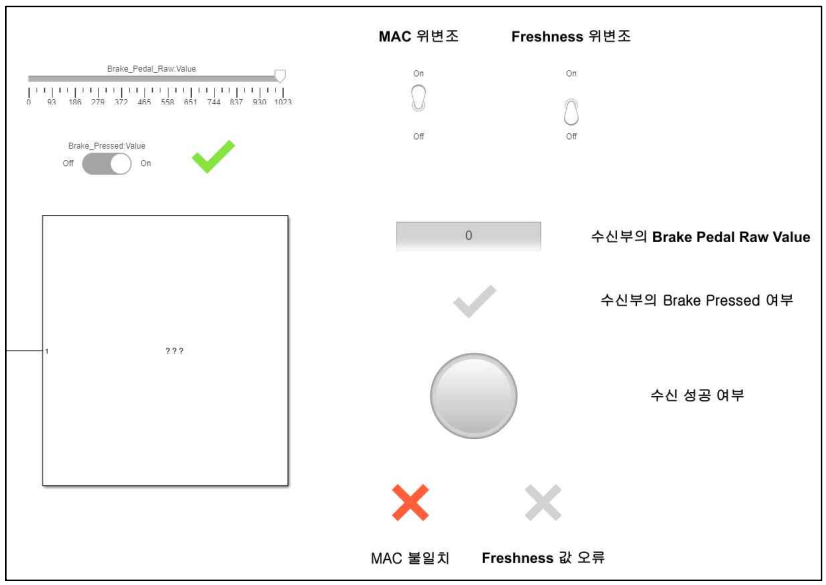


FIG32. 시뮬레이션 1

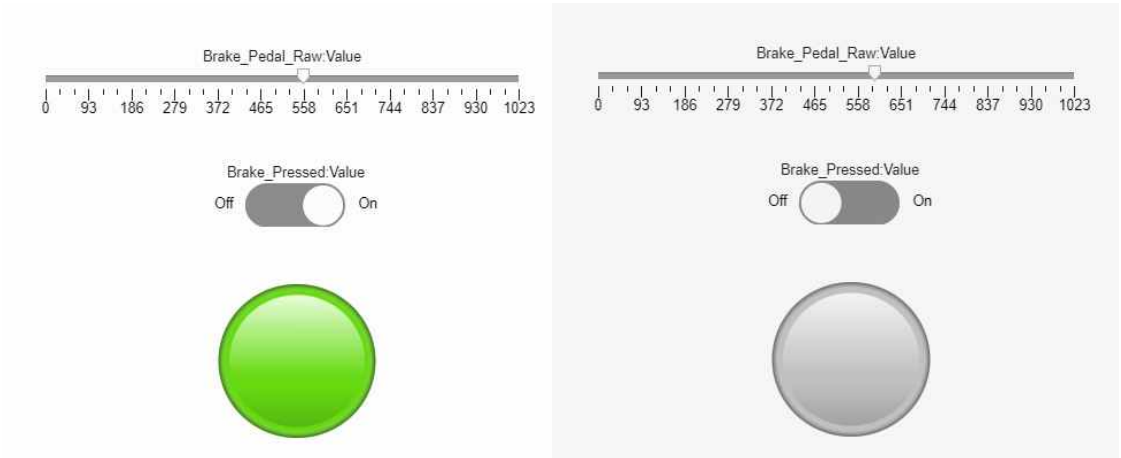


FIG33. 시뮬레이션 2

4.2. CAN-FD 기반 보안 통신 결과

(1) 데이터 무결성 확보:

HMAC-SHA256 알고리즘을 적용하여 AppData(3B)와 Freshness(4B)를 입력으로 MAC8을 생성하고, 수신부에서 동일한 과정을 통해 비교하였다. 결과적으로 위·변조된 데이터는 모두 차단되었으며, 데이터 무결성이 보장되었다.

(2) Freshness 검증 성능:

freshness_ok 함수는 단순한 비교·뺄셈 연산으로 동작하여 연산량이 적으면서도 재전송 공격을 효과적으로 탐지할 수 있었다. 단, Freshness 윈도우(W=1024) 설정에 따라 정상 데이터의 허용 범위와 보안 강도가 균형을 이룬다는 점을 확인하였다.

(3) Fail-Safe 동작:

MAC 또는 Freshness 검증 중 하나라도 실패하면, 수신 ECU는 데이터를 ECU 로직에 전달하지 않고 Lamp를 꺼버리는 Fail-Safe 구조가 구현되었다. 이를 통해 잠재적인 공격 상황에서 안전성을 확보할 수 있었다.

4.3. Ethernet 병렬 경로 확장 결과

본 프로젝트에서는 CAN-FD 외에도 **Ethernet(UDP 기반)** 전송 경로를 병렬로 추가하여 차세대 차량 네트워크 환경을 반영하였다.

(1) ETH_TX: 송신부에서 Raw, Pressed, Freshness, MAC8을 Outport로 전달받아 PDU(18B)를 구성한 후, UDP Send 블록을 통해 전송.

(2) ETH_RX: 수신부에서 UDP Receive로 데이터를 수신한 후, parsePDU 함수로 AppData, Freshness, MAC8을 분리하고 검증을 수행.

(3) 결과: CAN-FD와 동일한 보안 검증 로직을 적용할 수 있었으며, Ethernet 기반 통신 환경에서도 무결성 및 신선도 보장이 가능함을 확인하였다. 이는 CAN-FD와 Ethernet을 혼합하는 실제 차량 환경을 시뮬레이션한 결과로서 의미가 크다.

4.4. TLS 적용 가능성 검토

추가적으로, Ethernet 경로에 **TLS(Transport Layer Security)**를 적용하는 확장 가능성을 검토하였다.

(1) 보안 강도 강화: TLS를 적용하면 단순한 MAC 기반 무결성 검증을 넘어, 세션 키 교환·암호화·양방향 인증 기능을 제공할 수 있다.

(2) 오버헤드 문제: 다만 TLS는 세션 설정 및 암호화·복호화 과정에서 오버헤드가 크므로, 자원 제한적인 ECU에는 경량 TLS(Lightweight TLS) 혹은 DTLS와 같은 최적화된 방안이 필요하다.

(3) 본 프로젝트 의의: 본 연구에서는 MAC+Freshness 기반의 경량 검증을 중심으로 구현했으나, TLS 적용 가능성을 병행 검토함으로써 향후 자율주행 및 V2X 통신 환경까지 확장할 수 있는 토대를 마련하였다.

4.5. 시각화 및 UI 평가

(1) 시각화 요소: Lamp, Gauge, Switch, Indicator 등을 활용하여 보안 검증 결과를 직관적으로 확인할 수 있도록 설계하였다.

(2) MAC 위변조 vs Freshness 오류 구분: 빨간색 X, 회색 X, 초록색 체크 표시로 공격 유형과 정상 여부를 명확히 표현하였다.

(3) 실험적 의의: 사용자는 단순히 값 변화만 보는 것이 아니라, 보안 검증의 성공/실패를 한눈에 확인할 수 있었으며, 이는 교육적·실험적 효과가 크다.

4.6. 종합 평가

본 프로젝트의 결과를 종합하면 다음과 같다.

(1) CAN-FD 기반 보안성 검증: HMAC-SHA256과 Freshness Counter를 적용하여 데이터 무결성과 신선도를 동시에 보장하는 ECU 간 통신 모델을 구축하였다.

(2) Ethernet 병렬 확장: UDP 기반 Ethernet 전송을 병렬로 적용하여 CAN-FD의 제약을 보완하고, 차세대 차량 통신 환경을 반영하였다.

(3) TLS 확장 가능성 확인: 본 연구는 경량 보안 알고리즘 위주로 구현되었으나, Ethernet 기반 TLS 적용 가능성을 검토하여 향후 확장 방향을 제시하였다.

(4) Fail-Safe 구조 및 시각화 검증: 보안 검증에 실패한 데이터는 ECU 로직으로 전

달되지 않는 Fail-Safe 메커니즘을 구현하고, UI를 통해 결과를 직관적으로 확인하였다.

5. 결론 및 향후 연구 방향

5.1. 결론

본 연구는 MATLAB/Simulink 환경에서 **CAN-FD 및 Ethernet 기반 자동차 ECU 보안 통신 시스템**을 모델링하고, **HMAC-SHA256 기반 무결성 검증**과 **Freshness Counter 기반 재전송 공격 방지 기능**을 적용하여 보안성을 강화하는 방안을 실험적으로 검증하였다.

(1) 보안 강화된 CAN-FD 모델링

ㄱ. AppData, Freshness, MAC8을 포함한 24바이트 CAN-FD 페이로드를 설계·구현하였으며, 송·수신부에서 데이터 위·변조 및 재전송 공격을 효과적으로 탐지할 수 있었다.

ㄴ. freshness_ok 및 makeMAC 함수를 통해 데이터의 유효성을 이중으로 검증하고, 실패 시 Fail-Safe 메커니즘을 적용하였다.

(2) Ethernet 기반 확장

ㄱ. UDP 기반 Ethernet 송수신 경로를 병렬로 구현하여 CAN-FD의 제약을 보완하고, 차세대 차량 네트워크 환경(V2X, 클라우드 연계)을 반영하였다.

ㄴ. CAN-FD와 동일한 MAC/Freshness 검증 구조를 Ethernet 환경에서도 적용할 수 있음을 확인하였다.

(3) 시각화 및 직관적 검증 환경 구축

ㄱ. Lamp, Indicator, Gauge 등을 활용한 UI를 통해 정상/비정상 동작 여부를 직관적으로 확인할 수 있었으며, 이는 교육적·실험적 효과를 동시에 제공하였다.

이러한 성과는 실제 차량 통신 환경에서 발생할 수 있는 **데이터 위·변조, 스푸핑, 리플레이 공격에 대한 방어 가능성**을 보여주었으며, 자율주행 및 커넥티드카 환경에서 보안성을 높일 수 있는 근거를 제시하였다.

5.2. 향후 연구 방향

(1) TLS/DTLS 적용 연구

Ethernet 통신 구간에 TLS 또는 경량 DTLS를 적용하여 인증·암호화·세션 관리 기능까지 포함하는 고도화된 보안 프레임워크로 확장할 필요가 있다.

ECU 자원 제약 환경에서 TLS의 오버헤드를 최소화하기 위한 최적화 기법 연구가 요구된다.

(2) 다중 ECU 및 V2X 확장

본 연구는 송·수신 ECU 간 단일 경로에 집중했으나, 향후 다중 ECU 및 차량-인프라(V2I), 차량-차량(V2V) 통신 환경으로 확장할 필요가 있다.

이를 통해 실제 차량 네트워크 토폴로지와 유사한 시뮬레이션 검증이 가능할 것이다.

(3) 경량 보안 알고리즘 비교 연구

HMAC-SHA256 외에도 AES, SPECK, ECC 등 다양한 경량 보안 알고리즘을 적용·비교하여, 연산 속도·리소스 사용·보안 강도 간의 최적 균형점을 도출할 필요가 있다.

(4) 실차 검증 및 하드웨어 연계

Simulink 모델을 Hardware-in-the-Loop(HIL) 환경과 연계하여 실제 ECU, CAN/Ethernet 모듈, 보안 칩을 통한 실차 수준의 검증으로 확장해야 한다.

이를 통해 시뮬레이션 결과와 실제 차량 환경 간의 일치도를 평가하고, 실효성 있는 보안 대안을 도출할 수 있다.

(5) AI 기반 이상 탐지 연구

LLM이나 머신러닝 기반 이상 탐지 기법을 결합하여, 기존 MAC/Freshness 방식이 탐지하지 못하는 패턴(예: 지능형 공격, 희소 패킷 변조)을 탐지하는 하이브리드 보안 체계로 발전할 수 있다.

6. 개발 일정 및 역할 분담

1. 개발 일정

개발 일정은 프로젝트의 효율성과 마일스톤 준수를 위해 월별 및 주차별로 구성했다. 주요 마일스톤(착수보고서, 중간보고서, 최종보고서, 발표심사, 결과물 업로드)을 중심으로, 사전 기술 학습, Simulink 모델링, 보안 알고리즘 구현, UI 개발, 테스트 및 보고서 작성을 체계적으로 배치했다. 아래 표는 2025년 5월부터 10월까지의 일정을 나타낸다.

기간	주요 작업	세부 작업	담당자
5월 1-2주 (5.1-5.16)	착수보고서 작성 및 제출	- 프로젝트 개요 및 목표 정리 - CAN, TCP/IP, Simulink 사전 학습 계획 수립 - 착수보고서 작성 및 지도확인서 제출	홍재왕, 석재영, 레퐁푸
5월 3-4주 (5.17-5.31)	사전 기술 학습	- CAN 프로토콜 학습 (프레임 구조, 보안 취약점) - TCP/IP 및 Ethernet 학습 (V2X, 보안) - Simulink 기초 학습 (모델링, 라이브러리 활용)	홍재왕 (CAN) 석재영 (TCP/IP) 레퐁푸 (Simulink)
6월 1-4주 (6.1-6.30)	Simulink 환경 구축 및 초기 모델링	- Simulink 개발 환경 설정 (Vehicle Network Toolbox) - ECU, 센서 모델링 시작 - CAN 통신 시뮬레이션 구현 - Ethernet 통신 초기 시뮬레이션	레퐁푸 (환경 설정, 모델링) 홍재왕 (CAN 시뮬레이션) 석재영 (Ethernet 시뮬레이션
7월 1-2주 (7.1-7.18)	보안 알고리즘 설계 및 중간보고서	- 경량 보안 알고리즘(AES, SPECK) 설계 - Simulink에서 초기 알고리즘 통합 테스트 - 중간보고서 작성 및 중간평가표 제출	홍재왕, 석재영 (알고리즘 설계) 레퐁푸 (Simulink 통합, 보고서)
7월 3-4주 (7.19-7.31)	보안 알고리즘 구현	- CAN 통신에 메시지 인증(MAC) 구현 - Ethernet 통신에 TLS 적용 테스트 - 부품별 알고리즘 최적화 시작	홍재왕 (CAN 보안) 석재영 (Ethernet 보안) 레퐁푸 (부품 모델링 지원)
8월 1-4주 (8.1-8.31)	UI 개발 및 알고리즘 비교	- MATLAB App Designer로 UI 프로토타입 개발 - 보안 알고리즘 성능 비교(속도, 리소스, 보안성) - 로그 분석 LLM(LLaMA 3 8B) 파인튜닝 시작	레퐁푸 (UI 개발) 홍재왕, 석재영 (알고리즘 비교, LLM)
9월 1-2주 (9.1-9.19)	테스트 및 최종 보고서 작성	- Simulink에서 공격 시나리오(스푸핑, DoS) 테스트 - UI 완성 및 시각화 구현	홍재왕, 석재영, 레퐁푸

		- 최종보고서 및 최종평가표 작성	
9월 3-4주 (9.20-9.30)	발표 준비	- 테크위크 발표 자료 준비 - 시뮬레이션 및 UI 데모 시연 리허설	홍재왕, 석재영, 레퐁푸
10월 1-2주 (10.1-10.15)	결과물 업로드	- 최종 코드, 모델, 보고서 정리 - AWS S3에 결과물 업로드 - 프로젝트 문서화 완료	레퐁푸 (문서화) 홍재왕, 석재영 (검토

2. 역할 분담

구성원별 역할은 프로젝트 목표(자동차 전장부품 통신 프레임워크 개발 및 보안 알고리즘 검증)와 일정에 맞춰 다음과 같이 수정되었다:

가. 홍재왕 (CAN 프로토콜 담당): CAN 프로토콜의 메시지 구조와 취약점(스푸핑, 리플레이)을 분석하고, Simulink에서 CAN 통신 시뮬레이션을 구현한다. MAC, XOR, SHA-256을 적용해 유효 메시지 전송을 보장하며, 초기 구현을 완료한다. 보안 알고리즘 성능 비교(지연 시간, 메모리 사용)에 참여한다.

나. 석재영 (TCP/IP, Ethernet 담당): Automotive Ethernet과 TCP/IP의 통신 특성과 V2X 보안 요구사항을 분석한다. Simulink에서 Ethernet 통신 시뮬레이션을 구현하고, TLS 간소화 버전 적용을 테스트한다. 알고리즘 성능 비교와 데이터 흐름 분석을 지원한다.

다. 레퐁푸 (Simulink 담당): Simulink 환경을 구축하고, MathWorks 예제(Gauge, Slider)를 활용해 ECU, 센서, 계기판의 메시지 흐름을 모델링한다. 보안 알고리즘 통합 워크플로우를 설계하고, MATLAB App Designer로 UI(알고리즘 선택, 성능 시각화)를 개발한다. 보고서 작성과 문서화를 주도한다.

기존 역할에서 해시 알고리즘(SHA-256), 차량 통신 흐름(Gauge, Slider), 초기 CAN 보안 일부 구현 완료를 반영했다. LLM은 국가보안기술연구소 피드백에 따라 제외했으며, UI와 성능 비교를 구체화했다. 역할은 유연하게 조정 가능하다.

7. 멘토링 결과 반영 내용

7.1. 과제 목표의 명확화

착수보고서 대비 목표를 단순화하고 명확히 구분하라는 피드백을 반영하여, 본 과제의 목표를 다음 세 가지로 재정립하였다.

(1) Simulink 기반 보안 통신 환경 구축

(2) CAN 및 Ethernet 통신 시뮬레이션 구현

(3) CAN 프로토콜 보안성 강화 (HMAC, Freshness Counter 적용)

이를 통해 연구 범위가 지나치게 확장되는 것을 방지하고, 핵심적인 성과에 집중할 수 있도록 방향을 보완하였다.

7.2. 요구사항 및 제약사항 보완

멘토링에서 지적된 부분을 고려하여 요구사항과 제약사항을 보다 구체화하였다.

ㄱ. 네트워크 구성 요소 구분: CAN, Ethernet, 센서 세 부분을 명확히 나누어 문제점을 정의하였다.

ㄴ. 보안 강도와 알고리즘 적합성: AES-128 구현을 기준으로 하되, 8비트 환경에서 AES-256의 과도한 연산 부담을 분석하고, 이에 대한 대체 경량 암호 알고리즘을 후보군으로 제시하였다.

ㄷ. 알고리즘 선택의 합리화: 국제 표준화 문제로 채택에 한계가 있는 Speck 대신, 국내 개발 경량 블록암호인 CHAM 알고리즘을 검토 대상으로 반영하였다. 이를 통해 보안 강도와 실용성 사이의 균형을 확보할 수 있도록 하였다.

7.3. 설계 및 구현 상세화

초기 설계 대비 실용성을 강화하기 위해 다음과 같은 변경이 있었다.

ㄱ. **ECU 암호기능의 정상 동작 검증**을 최우선 과제로 설정하였다.

ㄴ. Simulink 모델 내 보안 모듈의 구조를 단순화하여 디버깅 가능성을 높이고, 실험적 오류의 원인을 추적할 수 있도록 하였다.

ㄷ. CAN과 Ethernet을 병렬적으로 구현하여 향후 보안 알고리즘 비교 및 확장 실험이 가능하도록 설계를 수정하였다.

7.4. 수행 인력 역할 분담의 구체화

멘토링에서 긍정적으로 평가받은 역할 분담 체계를 유지하되, 보다 세부적인 작업 책임을 반영하였다.

(1) 홍재왕: Ethernet 및 TLS 관련 통신 보안 기능 구현

(2) 석재영: CAN-FD 기반 보안 통신 및 Freshness Counter 검증

(3) 레퐁푸: UI 개발 및 보안 알고리즘 워크플로우 설계, 문서화 담당

이와 같이 팀원별 작업 범위를 명확히 구분함으로써, 중복 없이 효율적인 협업 체계를 구축하였다.

7.5. 향후 연구 방향

멘토링에서 제안된 다양한 경량 암호 알고리즘 적용 및 노하우 축적의 필요성을 반영하였다. 향후 연구에서는 CHAM, SPECK, ChaCha20-Poly1305 등 다양한 대안을 실험적으로 적용하여 보안성과 성능을 비교하고, 이를 기반으로 보안 ECU 기술의 상용화 가능성을 검토할 계획이다.

7. 참고 문헌

[1] R. Bosch GmbH, CAN Specification Version 2.0, Stuttgart, Germany, 1991.

[2] ISO 11898-1, Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling, International Organization for Standardization, Geneva, Switzerland, 2015.

[3] ISO 11898-7, Road vehicles — Controller area network (CAN) — Part 7: CAN FD data link layer specification, International Organization for Standardization, Geneva, Switzerland, 2015.

[4] MathWorks, Vehicle Network Toolbox Documentation: Simulink Support for CAN FD and Ethernet, MathWorks Inc., Natick, MA, USA, 2023. [Online]. Available: https://www.mathworks.com/help/vnt/

[5] M. Wolf, A. Weimerskirch, and C. Paar, "Security in Automotive Bus Systems," Proceedings of the Workshop on Embedded Security in Cars (ESCAR), pp.

1-12, 2004.

[6] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," Black Hat USA, pp. 1-91, Aug. 2015.

[7] FIPS PUB 180-4, Secure Hash Standard (SHS), Federal Information Processing Standards Publication, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 2015.

[8] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, Internet Engineering Task Force (IETF), Feb. 1997.

[9] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Springer, 2011.

[10] MathWorks, MATLAB Function Blocks and Code Generation for Embedded Systems, MathWorks Inc., Natick, MA, USA, 2023. [Online]. Available: https://www.mathworks.com/help/simulink/ug/matlab-function-block.html